Array.sort() หลักการทำงานของ sort() คือ จะแปลงค่าภายใน Array ให้เป็น String จากนั้นจะทำการเปรียบเทียบค่าโดยใช้ UTF-16 ซึ่งจะ Return ออกมาเป็น Array

Syntax

firstEl คือ Element ตัวแรกที่ต้องการเปรียบเทียบ

secondEl คือ Element ตัวที่สองที่ต้องการเปรียบเทียบ

- **`sort ()`**
- **`sort (compareFunction)`**

Syntax Arrow Function

- **`sort((firstEl, secondEl) => { ... } )`**
- **`sort(function compareFn (firstEl, secondEl) => { ... } )`**

Syntax : sort ()

Primitive Type

ตัวอย่างที่ 1 : Integer

ยกตัวอย่างโดยกำหนดค่าภายใน Array เป็น Primitive Type

```
let arrInteger = [5, 1, 3, 4, 6, 8] // Array ที่มี Type เป็น Integer
arrInteger.sort();
```

ผลลัพธ์ที่ได้

```
console.log(arrInteger); //  Output [1, 3, 4, 5, 6, 8]
```

```
[ 1, 3, 4, 5, 6, 8 ]
```

ตัวอย่างที่ 2 : Integer

ยกตัวอย่างโดยกำหนดค่าภายใน Array เป็น Primitive Type

```
let arrInteger = [5, 1, 13, 34, 6, 8] // Array ที่มี Type เป็น Integer
arrInteger.sort();
```

ผลลัพธ์ที่ได้

```
console.log(arrInteger); //  Output [1, 13, 34, 5, 6, 8]
```

```
[ 1, 13, 34, 5, 6, 8 ]
```

HINT : จะสังเกตได้ว่าลำดับในการเรียงเลขไม่ได้เรียงแบบธรรมชาติ เป็นเพราะว่าในการ sort ครั้งนี้มันไม่ได้มองว่าเป็นตัวเลขแต่มองว่าเป็น String โดยเปรียบเทียบตาม UTF-1

ตัวอย่างที่ 3 : String

ยกตัวอย่างโดยกำหนดค่าภายใน Array เป็น Primitive Type

```
let arrString = ['Jan', 'Feb', 'Dec', 'March', 'April'] // Array ที่มี Type เป็น String
arrString.sort();
```

ผลลัพธ์ที่ได้

```
[ 'April', 'Dec', 'Feb', 'Jan', 'March' ]
```

```
console.log(arrString); // Output [ 'April', 'Dec', 'Feb', 'Jan', 'March' ]
```

Syntax : sort (compareFunction)

ตัวอย่างที่ 1 : Integer

ยกตัวอย่างโดยกำหนดค่าภายใน Array เป็น Primitive Type

```
let arrInteger = [5, 1, 13, 34, 6, 8] // Array ที่มี Type เป็น Integer

arrInteger.sort(function (a,b){
    return a-b;
})
```

ผลลัพธ์ที่ได้

```
[ 1, 5, 6, 8, 13, 34 ]
```

```
console.log(arrInteger); // Output [ 1, 5, 6, 8, 13, 34 ]
```

Arrow Function : `arrInteger.sort() = (a,b) => a-b`

ตัวอย่างที่ 2 : String

ยกตัวอย่างโดยกำหนดค่าภายใน Array เป็น Primitive Type

```
let names = ['Mary','Alice','Peter','Bob'];
function compareName(a,b){
  if(a < b) {return -1;}
  if(b > a) {return 1;}
  return 0;
}
names.sort(compareName);
```

ผลลัพธ์ที่ได้

```
[ 'Alice', 'Bob', 'Mary', 'Peter' ]
```

ตัวอย่างที่ 3 : Object

```
let player = [ {name:"Jojo", id:40000},
        {name:"Pravut", id:8400},
        { (property) name: string
        {name:"Meow", id:9},
        {name:"Oreo", id:10},
        {name:"Jojo", id:15000} ];

function compareName(a,b){
  return a.name.localeCompare(b.name);
}
player.sort(compareName);
```

ผลลัพธ์ที่ได้

```
[
  { name: 'banglee', id: 69 },
  { name: 'Jojo', id: 40000 },
  { name: 'Jojo', id: 15000 },
  { name: 'Meow', id: 9 },
  { name: 'Oreo', id: 10 },
  { name: 'Prayut', id: 8400 }
]
```

Syntax

- **sort((firstEl, secondEl) => { ... })**
- **sort(function compareFn (firstEl, secondEl) => { ... })**

Object

ตัวอย่างข้อมูล

```
let player = [ {name:"Jojo", id:40000},
{name:"Prayut", id:8400},
{name:"banglee", id:69},
{name:"Meow", id:9},
{name:"Oreo", id:10},
{name:"Jojo", id:15000} ];
```

ตัวอย่างที่ 1 : Object

ทดลองเขียนตาม Syntax

```
player.sort();
```

ผลลัพธ์ที่ได้

```
console.log(player);
```

```
[
  { name: 'Jojo', id: 40000 },
  { name: 'Prayut', id: 8400 },
  { name: 'banglee', id: 69 },
  { name: 'Meow', id: 9 },
  { name: 'Oreo', id: 10 },
  { name: 'Jojo', id: 15000 }
]
```

จะเห็นได้ว่า output ที่ออกมาไม่ได้มีการ sort ข้อมูล เพราะเราจะต้องทำการระบุ Key ที่ต้องการ Sort ก่อน

ตัวอย่างที่ 2 : Object (Keys : name)

เราสามารถ sort() Object ได้โดยการเจาะจงไปที่ Keys ที่เราต้องการ sort() ตัวอย่างเช่น

การจัดลำดับของข้อมูลโดยเรียงตามตัวอักษร (Key : name) และสามารถนำฟังก์ชันอื่นเข้ามาช่วยในการ sort ข้อมูลได้ ตัวอย่างเช่น localeCompare()

```
player.sort(function (a, b) {
    return a.name.localeCompare(b.name);
});
```

localeCompare คือ prototype function ที่มีอยู่แล้วในตัวแปรประเภท string ของ javascript ซึ่ง function นี้เอาไว้สำหรับเปรียบเทียบระหว่าง string 2 ตัวว่าตัวไหนมาก่อนตามหลักของแต่ละภาษา

Arrow Function

```
player.sort((a,b) = > a.name.localeCompare(b.name));
```

ผลลัพธ์ที่ได้

```
console.log(player);
```

```
[
  { name: 'Jojo', id: 40000 },
  { name: 'Prayut', id: 8400 },
  { name: 'banglee', id: 69 },
  { name: 'Meow', id: 9 },
  { name: 'Oreo', id: 10 },
  { name: 'Jojo', id: 15000 }
]
```

ตัวอย่างที่ 3 : Object (Keys : name)

เราสามารถ sort() Object ได้โดยการเจาะจงไปที่ Keys ที่เราต้องการ sort() ตัวอย่างเช่น

การจัดลำดับของข้อมูลโดยเรียงตามตัวอักษร (Key : name)

```
player.sort(function(a, b) {
    let nameA = a.name.toUpperCase(); // สร้างตัวแปรเพื่อรับค่า Parameter ที่รับมาแปลงให้เป็นตัวพิมพ์ใหญ่ทั้งหมด
    let nameB = b.name.toUpperCase(); // สร้างตัวแปรเพื่อรับค่า Parameter ที่รับมาแปลงให้เป็นตัวพิมพ์ใหญ่ทั้งหมด
    if (nameA < nameB) { // return -1; ถ้าเปรียบเทียบ UTF-16 แล้ว A<B  A จะขยับไปอยู่ทางขวาของ B
    } if (nameA > nameB) { // return 1;ถ้าเปรียบเทียบ UTF-16 แล้ว A<B  A จะขยับไปอยู่ทางซ้ายของ B
    } return 0; }); // ถ้า A = B จะอยู่ที่เดิม
```

ผลลัพธ์ที่ได้

```
console.log(player)
```

```
[
  { name: 'Jojo', id: 40000 },
  { name: 'Prayut', id: 8400 },
  { name: 'banglee', id: 69 },
  { name: 'Meow', id: 9 },
  { name: 'Oreo', id: 10 },
  { name: 'Jojo', id: 15000 }
]
```

ตัวอย่างที่ 4 : Object (Keys : id)

```
player.sort(function (a,b) { return a.id - b.id; })
```

Arrow Function

```
player.sort((a,b) => a.id - b.id );
```

ผลลัพธ์ที่ได้

```
console.log(player)
```

```
[
  { name: 'Meow', id: 9 },
  { name: 'Oreo', id: 10 },
  { name: 'banglee', id: 69 },
  { name: 'Prayut', id: 8400 },
  { name: 'Jojo', id: 15000 },
  { name: 'Jojo', id: 40000 }
]
```

findIndex มีหน้าที่ return เลข index ของ array ตัวแรกที่ผ่าน testing function แต่ถ้าไม่เจอจะ return ค่า -1 ออกมา (ถ้าพบค่าแล้วจะหยุดทำทันที)

Syntax

```
array.findIndex(function(currentValue, index, arr), thisValue)
```

- **currentValue** คือ Element ปัจจุบันใน Array ที่กำลังประมวลผล
- **index** คือ Index ของ Element ที่กำลังประมวลผลอยู่ในปัจจุบัน
- **arr หรือ Array** คือ Array ปัจจุบันที่ findIndex เรียกใช้
- **thisValue คือ** argument ที่ส่งผ่านไปหา Function เพื่อเป็นค่า This ซึ่งถ้าเป็นค่าว่างจะส่ง undefined แต่ถ้าพบค่าจะส่งค่า This ใช้สำหรับ CallbackFn

Arrow function

```
findIndex((element) => { ... } )
findIndex((element, index) => { ... } )
findIndex((element, index, array) => { ... } )
```

Primitive Type

ตัวอย่างที่ 1 : Integer

```
let ages = [3, 10, 18, 20]; // สร้าง Array ที่มี Element เป็น Integer
function checkAge(age) {
  return age > 18; // สร้าง function ขึ้นมา โดยให้ Return age > 18
}
```

Arrow Function

```
let checkAge = ages.findIndex(age => age > 18);
```

ผลลัพธ์ที่ได้

```
index : 3
```

```
console.log(`index : ${ages.findIndex(checkAge)}`) // Returns 3 (Index ช่องที่ 3 จากใน Array)
// เรียกใช้งาน finIndex โดย Function รับค่า Array ที่มี Element เป็น Integer
```

ตัวอย่างที่ 2 : Integer

```
let ranks = [1, 5, 7, 8, 10, 7]; // สร้าง Array ที่มี Element เป็น Integer
let index = ranks.findIndex(function (rank){
    return rank === 7; // สร้าง function ขึ้นมา โดยให้ Return rank ที่มี DataType เดียวกันและมีค่าเท่ากับ 7
})
```

Arrow Function

```
let index = ranks.findIndex(rank => rank === 7);
```

ผลลัพธ์ที่ได้

```
index : 2
```

```
console.log(`index : ${index}`); // Return 2 (Index ช่องที่ 2 จากใน Array)
```

ตัวอย่างที่ 3 : String

```
let str = ['Hello','Hi','Morning']; // สร้าง Array ที่มี Element เป็น String
function checkStr (str,index){
    return str === 'Hi' && index > 1;
    // สร้าง function ขึ้นมา โดยให้ Return str ที่มี DataType เดียวกันและมีค่าเท่ากับ Hi และ มีค่า Index > 1
}
```

Arrow Function

```
let checkStr = str.findIndex((str,index) => str === 'Hi' &&
index > 1)
```

ผลลัพธ์ที่ได้

```
index : -1
```

```
console.log(str.findIndex(checkStr)); // Return -1
```

จะเห็นว่าผลลัพธ์ที่ return ออกมาคือ -1 (ไม่พบค่า) สาเหตุก็คือ ในการทำ findIndex มันได้พบค่าของ ‘Hi’ แต่มีอีกเงื่อนไขคือค่า Index ต้องมากกว่า 1 จึงทำให้ไม่พบค่าของข้อมูล

ตัวอย่างที่ 4 : String

```
let str = ['Hello','Hi','Morning']; // สร้าง Array ที่มี Element เป็น String
function checkStr (str,index,arr){
    console.log(arr); // ให้แสดง Array ที่เข้ามา Test Function
}
```

Arrow Function

```
let checkStr = str.findIndex((str,index,arr) => console.log(arr))
```

ผลลัพธ์ที่ได้

```
console.log(str.findIndex(checkStr)); //จะแสดง Array ตามจำนวนข้อมูลภายใน Array
```

```
[ 'Hello', 'Hi', 'Morning' ]
[ 'Hello', 'Hi', 'Morning' ]
[ 'Hello', 'Hi', 'Morning' ]
```

Object

ตัวอย่างข้อมูล

```
const result = [
    {name:'John', grade: 'A', id: 1001 },
    {name:'Ben', grade: 'C', id: 1002 },
    {name:'Anthony', grade: 'B', id: 1003 },
    {name:'Tim', grade: 'B', id: 1004 }
  ]
```

ตัวอย่างที่ 1 : Object (Key : id)

```
let checkResult = result.findIndex(function (value){
    return value.id % 2 == 0
// รับ result ที่เป็น Object เข้ามาเพื่อหาว่า Object.value ตัวไหนที่ Mod 2 เท่ากับ 0
})
```

Arrow Function

```
let CheckResult = result.findIndex(value => value.id % 2 == 0)
```

ผลลัพธ์ที่ได้

```
index : 1
```

```
console.log(checkResult) // Return Index ช่องที่ 1
```

ตัวอย่างที่ 2 : Object (Key : grade)

```
let checkResult = result.findIndex(function (value){
    return value.grade === 'B'
// รับ result ที่เป็น Object เข้ามาเพื่อหาว่า Object.grade ตัวไหนที่มีค่าเท่ากับ B
})
```

Arrow Function

```
let CheckResult = result.findIndex(value => value.grade === 'B')
```

ผลลัพธ์ที่ได้

```
index : 2
```

```
console.log(checkResult) // Return Index ช่องที่  2
```

ตัวอย่างที่ 3 : Object (Key : name)

```
let checkResult = result.findIndex(function (value,index){
    return value.name === 'Tim' && index != 0
// รับ result ที่เป็น Object เข้ามาเพื่อหาว่า Object.name ตัวไหนที่มีค่าเท่ากับ Tim และ index ไม่เท่ากับ 0
})
```

Arrow Function

```
let CheckResult = result.findIndex((value,index) =>
value.name === 'Tim' && index != 0)
```

ผลลัพธ์ที่ได้

```
index : 3
```

```
console.log(checkResult) // Return Index ช่องที่ 3
```

GITHUB : https://github.com/Runlertjit/INT201-G09-GroupWorks-04.git